Application of Focus + Context to UML

Benjamin Musial **Timothy Jacobs**

Department of Electrical and Computer Engineering
Air Force Institute of Technology
Dayton, Ohio, USA 45430

{Benjamin.Musial, Timothy.Jacobs}@afit.edu

Abstract

UML class diagrams of complex software systems are frequently very large, making it difficult to gain a detailed understanding of the underlying software components within the context of the overall software system. To overcome this limitation, we apply focus + context visualization techniques to an interactive UML browser. We display software components at varying levels of detail according to a dynamic degree of interest function. We then lay out the diagram components to maximize space efficiency while emphasizing the relationships between components. Preliminary results of our work are presented here.

1 Introduction

Software systems are often very large and complex with large numbers of interconnected components. It is difficult for humans to interpret software in its executable form; thus, a variety of abstractions, such as classes and methods, have been established to facilitate specification and understanding of software. These abstractions may be modelled textually (programming languages), mathematically (formal specifications), or visually (visual modelling languages). Due to the size and complexity of many software systems, and the increased bandwidth the human visual system provides for processing information, visual modelling languages have become the preferred method for specifying, analysing, and communicating the structure and behavior of software systems. For object-oriented software systems, the Unified Modelling Language (UML) is currently the industry standard.

Even with the increased bandwidth of a visual language such as UML, the size of many software systems still leads to visual specifications that are extremely large and complicated. For just a moderately sized system, a typical UML class diagram covers many pages and is too large for efficient knowledge acquisition of the overall system structure and lower level details. Common solutions to this problem are multi-page printouts, overview windows for navigating the entire diagram, and hypertext links between related sub-models. Multi-page printouts can effectively show overall system structure and individual component details; however, they suffer from many drawbacks such as increased visual scanning, difficult production and management, and lack of support for interaction and dynamic editing. Overview windows and hypertext may provide access to overall system structures and individual component details, but this does not occur in a single view, thus requiring more effort and reducing the effectiveness of the visual representation.

We propose another approach to this problem that applies focus + context techniques to an interactive UML diagram editing tool. Focusing on the UML class diagram, we define multiple levels of detail representations for classes and relationships. We display an appropriate level of detail representation for each component using a degree of interest function based on frequency of access and distance. Finally, we smoothly animate the relocation of software components within the diagram to emphasize hierarchical relationships while maximizing space economy.

2 Visual Modelling Requirements

Over the past decade UML has become the standard modeling language for software systems. UML provides diagrams to model static software structure and different aspects of dynamic system behavior (UML 2000). The symbols and associated semantics are well known within the software industry. To be accepted in the software community, any visualization solution for software systems must incorporate UML symbology and semantics.

The class diagram is the most widely used of all UML diagrams (Brodsky and Cook, 1999). A class diagram includes rectangular nodes depicting classes with annotated lines between these nodes to indicate the relationships between classes. Rectangular nodes depicting classes are annotated with textual labels to identify the class and its associated state variables and behaviors. It is not uncommon for a software system to be composed of hundreds or thousands of unique classes. Analysis and comprehension of such a software system requires both a high level overview of the system structure (consisting of numerous classes and the relationships among them) and a detailed examination of the characteristics of individual classes or small subsets of classes. Software visualization techniques should provide access to both high level and detailed views.

Paige identifies nine factors to consider when designing and evaluating modelling languages—simplicity, uniqueness, consistency, seamlessness, reversibility, scalability, supportability, reliability, and space economy (Paige, Ostroff, Brooke 2000). One could argue the degree to which UML satisfies many of these factors; however, we consider only the issue of space economy.

Space economy requires that "models should take up as little space on the printed page as possible (Paige, Ostroff, Brooke, 2000)." UML does not effectively address space economy, leaving lots of unused space within and between components.

As a consequence of UML's inefficient use of space, UML requires many pages to model large systems. Navigating through many pages significantly increases the time to access components of interest. In addition, by spreading the model over multiple pages, it is not possible to simultaneously view high level system structure and individual component details. The goal of this work is to maintain the symbology and semantics of UML while improving its space efficiency to accommodate rapid access to both high level and detailed system information.

3 Related Work

Card *et al* (1999) use the terminology "focus + context" when referring to visual techniques that provide simultaneous access to both overview and detailed information. According to Card, focus + context is based on the following three premises:

1. The user requires both the overall picture and a detailed view.
2. There may be different requirements for the information in the detailed view than in the overall picture.
3. Both displays may be combined into a single dynamic view.

Bertin (1977) explains that it is beneficial to combine the two views because when information is separated into two regions, visual search and working memory limitations result in reduced performance. A second reason to combine the two views was identified by Furnas (1981) during research into fisheye views. It showed that the user's attention drops off away from the areas of detail. Various methods take advantage of this observation, relating the level of detail in a region to the apparent interest in the region by the user. This improves the space-time efficiency of the user, which is a limited resource. This efficiency is increased as more useful information is displayed per unit area and the average time to find an item of interest is reduced as it is more likely to be already displayed on the screen. Card *et al* (1999) further explain how focus + context uses these principles to reduce the cost of accessing information which in turn results in increased cognition.

Focus + context can be implemented with a variety of methods that provide selective reduction of the presented information. These methods include filtering out extraneous information, selective aggregation of related information, micro-macro readings, highlighting, and distortion (Card, Mackinlay, Schneiderman, 1999)

Of particular interest for graph structures is the concept of selective aggregation. This method hides aggregate elements within another component. To achieve a focus + context view, hierarchical structures away from the user's focus are collapsed. When content is brought into focus, the hierarchy expands revealing the aggregate relationships within. UML has aggregation and inheritance relationships that are ideal for this technique.

Other transforms may be required to blend the detailed display with the global view. A variety of fisheye lenses are available for this purpose (Leung and Apperley, 1994). In order to preserve the appearance of links and location context in UML, a lens that minimizes distortion is best for this application. As such, a step-wise function similar to that used in the Furnas Fisheye (Furnas, 1981) is more suitable than nonlinear or polar transforms.

Köth and Minas (2001) apply focus + context techniques to UML models in DIAGEN. DIAGEN is designed to provide adjustable detail levels for editing large diagrams. When used for UML diagrams, DIAGEN uses selective aggregation to hide the contents of packages and other components to reduce the amount of detailed information displayed. While package aggregation can improve navigation and access to system level structures, it does not address the structures within packages which can still be considerably large and difficult to navigate.

With any model, some contextual awareness will be developed by the user as use increases. During the transition to a more efficient layout, it is highly desirable to maintain this context. Along these lines, Wickens (1992) develops the concept of visual momentum to create a set of user interface design principles. One of these principles suggests that to preserve context, smooth transitions should be used so that the relative position of each element may be tracked.

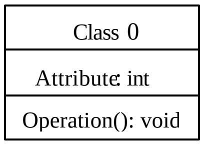
4 Methodology

In this work, we apply the concepts of focus + context to an existing, open source, UML editor known as ArgoUML. We develop a fisheye lens that displays less detail for components with a smaller degree of interest and we apply selective aggregation techniques to hide components that are beyond a specified degree of interest. Finally, we apply a graph layout algorithm that arranges components to emphasize hierarchical relationships while maximizing space efficiency.

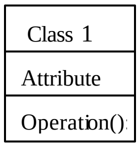
ArgoUML is a graphical UML editor that facilitates the specification of UML diagrams for modelling software systems (ArgoUML, 2002). ArgoUML is derived from the Graph Editing Framework (GEF) that provides basic capabilities to display a variety of simple shapes and connect them with links (GEF, 2002). In this research we modify the UML representations in ArgoUML and graphical layout algorithms of GEF to implement our focus + context techniques.

We use selective filtering and modified graphical elements to create multiple level of detail (LOD) representations for UML classes. At the highest level of detail, each class consists of a rectangle divided into three segments that include textual labels for the class name, attributes, and methods. At lower levels of detail, the textual labels are selectively filtered according to the perceived relevance of the information. Although other possibilities for aggregation and filtering may exist, our solution supports six levels of detail as summarized here:

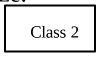
- LOD 0 contains the highest amount of detail. For a UML class representation this includes a full size graphical representation that includes textual labels for all attributes and methods along withtype information.



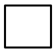
- LOD 1 hides attribute and return types while minimizing margin size.



- LOD 2 only displays the name of the class at a reduced font size.



- LOD 3 removes all textual information and only indicates the presence of a component.



- LOD 4 contains no textual information and indicates the presence of a component at a reduced size.

- LOD 5 completely hides the component from the user.

A class is displayed at a particular level of detail using a degree of interest (DOI) function based on the frequency of access to a particular class and its distance from the current class in focus (Equation 1).

(1) $DOI \propto \log_2(freq) + (max\_visible\_doi - dist)$

where *freq* is the number of times the node in question has been accessed by the user;

dist is the graphical distance from the node to the focal point in the diagram; and

max_visible_doi is the maximum DOI at which a node is visible.

Each time a node is accessed, the degree of interest is recalculated and the display is updated to reflect the appropriate level of detail for each node. Should a displayed node be further from the focal point than a hidden node, all hidden nodes along the path of associations between the two nodes have their LOD increased to 4 to prevent dissection of the graph. If a cycle exists in the graph, the level of detail is determined by the longest path from the node in focus.

To demonstrate this mapping of LOD representations to the DOI function, consider the sample class diagram in Figure 1. If “Class 0” is selected as the node of focus, the representation of all remaining classes is modified as shown in Figure 2. “Class 0” is selected as the node of focus so it is displayed with no change at LOD 0. The degree of interest for “Class 1” is one lower than that for “Class 0” so “Class 1” is displayed at LOD 1. As each subsequent class in this diagram is one additional graphical link away from the node of focus, “Class 0”, each subsequent class is represented at the next lower level of detail. Beyond “Class 4” no classes in this path would be shown.

Generalization and *aggregation* are key concepts in object-oriented software systems. These relationships are hierarchical, with a generalization relationship linking a more general class to one or more classes that inherit attributes of the general class. Similarly, aggregation links a container class to one or more classes representing parts that are combined within the container class. As hierarchical relationships, both generalization and aggregation are good candidates for selective aggregation techniques that hide lower levels of the hierarchy. For instance, there may be little value in showing that a car consists of wheels, an engine and chassis, when the user is only interested at the time in seeing that there is a car. UML also includes association relationships that indicate a non-hierarchical relationship between two classes.

We apply selective aggregation, based on UML’s association, aggregation, and inheritance relationships, for those classes that are graphically distant from the class in focus. To indicate that classes are hidden, we continue to display the appropriate UML symbology for inheritance or aggregation. For an association relationship we simply display a partial link indicating hidden classes. When the content is brought into focus, the hidden hierarchy expands revealing the aggregate relationships within.

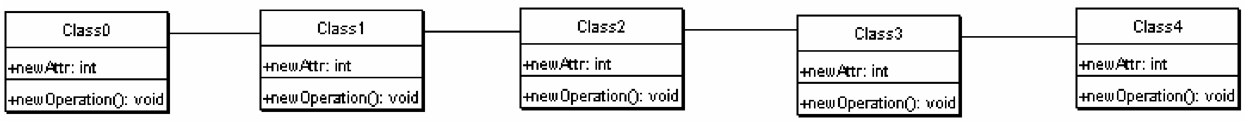


Figure 1. Class diagram prior to application of visualization techniques.

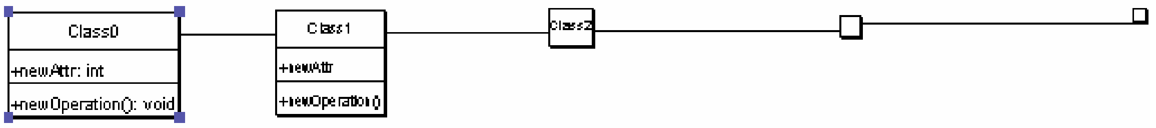


Figure 2. Class diagram following application of focus + context techniques.

Figures 3a and 3b show the results of our selective aggregation techniques for a simple class diagram. Figure 3a depicts a class diagram before selective aggregation is applied. The link with a triangle at the end depicts an inheritance relationship and the link with a diamond depicts an aggregation relationship. The triangle and the diamond are positioned at the end of the link associated with the parent class. Figure 3b presents the class diagram after applying the selective aggregation techniques. The triangle, diamond, and a small line continue to be displayed to indicate that the hidden classes are connected via some relationship to the class. Any classes of a lower DOI and further from the focal point are also hidden from view. It is expected that if the aggregate class is of little interest and is thereby hidden, then so will those objects that it is associated with.

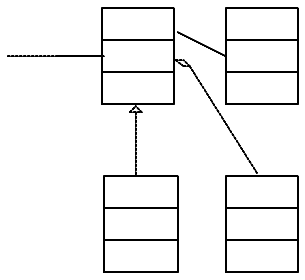


Figure 3a Part of class diagram.

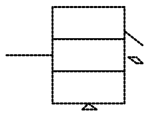


Figure 3b. Class diagram after focus + context showing relationships.

To maximize the space efficiency of our focus + context techniques for UML, the positions of elements in the model must also be modified. In evaluating how best to accomplish this, a variety of UML graph layout algorithms were examined (Auburn, 2002; Rational, 2002). A recurring theme in these algorithms is the presentation of hierarchical inheritance relationships down the vertical axis. As this is a commonly used approach and is understood and expected by users, this presentation approach is enforced in our layout algorithms. Although a variety of other factors such as edge crossings and connection points can impact the graph layout, our algorithm emphasizes hierarchical relationships and improved space efficiency.

Application of our layout algorithm requires the movement of nodes and links in the class diagram. If done instantaneously, this movement may cause the user to lose the original context. To avoid this, we use smooth animation when moving graphical elements between their original and revised positions. We also move the elements in two stages. The first stage positions elements in the appropriate level of the inheritance hierarchy. The second stage positions leaf nodes to optimize space efficiency. The graph layout algorithm is applied after all degree of interest and selective aggregation functions.

An explanation and example of the results of applying the visualization features and layout algorithms follows. The initial class diagram for this example is shown in figure 4.

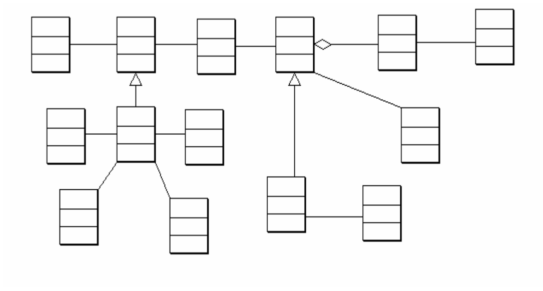


Figure 4. Initial class diagram.

The focus + context techniques are enabled through a context sensitive menu. The results of setting the upper left node of figure 4 as the focal point are presented in figure 5. Selective aggregation can be seen for both aggregation and inheritance relationships from the far right node.

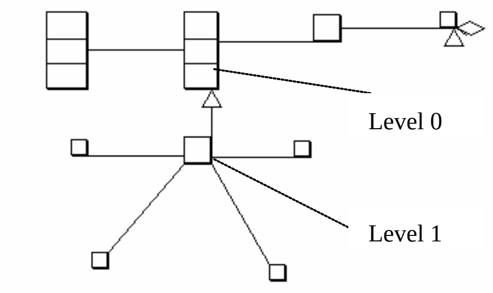


Figure 5. Class diagram with focus + context applied.

Prior to applying the layout algorithm, the hierarchical level of each visible node is determined. A difference of one level exists across an inheritance relationship. Each node connected via an association or aggregation relationship will have the identical level. If a cycle exists, a node may be connected to multiple hierarchical levels. In this case, a node that is a direct descendant of an inheritance relationship is placed at a lower level. All other nodes in the cycle remain at the original level determined through the association relationships. All nodes having the same level are placed on the same row. Figure 5 identifies the level (and row number) of the nodes on each side of the inheritance relationship.

The first stage of the layout algorithm examines each row (starting from the top row) and searches for generalization links from each of the nodes in that row. Figure 5 contains one node with a generalization link. The *x* position of a node with generalization links is equal to the average of the *x* positions of the generalized classes in the row above it. Should this position not be free it is placed as close as possible to the desired position on the same row.

All remaining nodes on a row without a laid-out position are placed in order based on their x position in the row prior to the layout process. Each node in a row is separated by a horizontal gap of a fixed distance. Similarly the y position for each row is determined by adding a fixed size gap to the lowest point in the above row. This stage of the algorithm has the effect of shrinking the overall display area yet it maintains the general layout of the original diagram. The results of this process are shown in figure 6.

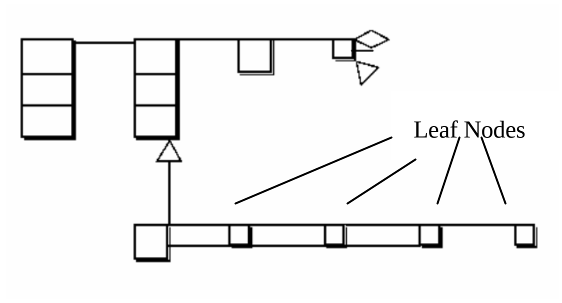


Figure 6. Class diagram after phase 1 of layout algorithm.

The second stage of the layout algorithm searches each row for nodes with leaf nodes on the same row. If these leaf nodes have a relatively low LOD (and vertical height), they are shifted to the right side of the associated node. The leaf nodes are arranged radially with the radius proportional to the number of leaf nodes. If possible, the position of other nodes on that row is left unchanged. Should there be insufficient room, then all nodes to the right of the leaf nodes are shifted right to establish a suitable gap.

The bottom left node of figure 6 has four associated leaf nodes on its row; these are rearranged in a radial fashion by the second stage of the algorithm as shown by figure 7. This part of the algorithm allows a greater number of nodes to be displayed than otherwise possible while still preserving user context. This algorithm is particularly effective when placing leaf nodes with a low LOD around a node with a high LOD.

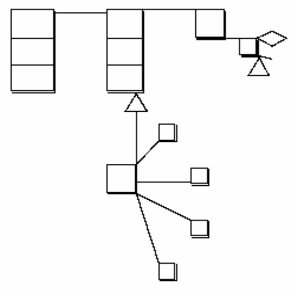


Figure 7. Class diagram after phase 2 of layout algorithm.

5 Analysis

To analyse the effectiveness of our focus + context techniques we compare our UML displays to those in the original ArgoUML. The overall goal of our work is to maintain the symbology and semantics of UML while improving its space efficiency to accommodate rapid access to both high level and detailed system information. We analyse the effectiveness of our techniques with respect to this goal.

If we assume fairly simple classes such as those in our earlier examples, then we can fit roughly 24 classes on a single screen. By applying our degree of interest and graph layout algorithms, we can display approximately 75 classes on a single screen. Thus we obtain three times the space efficiency without even resorting to selective aggregation. The degree to which selective aggregation increases our space efficiency is dependent on the level of fan out in the system relationships.

Access to high level and detailed system information is more difficult to evaluate. With original representations, we will need to scroll through multiple pages if we have more than 24 classes. In order to deal with large systems, the user would have to commit much of the overall system information to long term memory. By applying our focus + context techniques, we can readily view 75 classes simultaneously with the structure of hundreds of classes potentially available using selective aggregation. This enables the user to offload more cognitive processing to the external visual presentation.

Using original ArgoUML representations, details are readily available for up to 24 classes at a time; however, the user can still only focus on one area of the screen at a time. With our focus + context view, only a few components are in focus at any given time. The user needs to interactively select another component to bring it into focus. Once the appropriate component is brought into focus, the time to access detailed information is similar to that of the original UML. If all components that the user is likely to access are displayed on the same page, it is likely to be faster to switch focus using the original UML; however, for the common case where components are spread over multiple pages in the original UML, our focus + context techniques are likely to improve access time since more components are accessible from the same screen.

6 Conclusion

Focus + context techniques have proven effective for understanding large quantities of information. We have applied two such techniques, a fisheye lens and selective aggregation, to the presentation of UML diagrams for software systems. We maintain the symbology and semantics of the UML while presenting more information in the same amount of space. This improves access to that information, thereby enabling the user to take in more information in the same amount of time. To improve user understanding of the underlying software system, we maintain access to both high level structural infomation as well as individual component details. Furthermore, we provide a graph layout algorithm that organizes software components to emphasize hierarchical relationships while improving space efficiency.

7 Acknowledgements

This work is funded in part by the United States Air Force Office of Scientific Research. The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

8 References

ARGOUML.TIGRIS.ORG (2002): ArgoUML User Manual, http://argouml.tigris.org/documentation/ defaulthtml/manual/.

AUBURN UNIVERSITY (2002): Graphical Representations of Algorithms, Structures, and Processes (JGRASP), http://www.eng.auburn.edu/ grasp/.

.BERTIN, J (1977): *Graphics and Graphic Information Processing*, Berlin, deGruyter Press.

BRODSKY, S., COOK, S. (1999): OMG Analysis & Design PTF UML 2.0 Request for Information Response from IBM, http://cgi.omg.org/docs/ad/99-12-08.pdf

CARD, S., MACKINLAY, J., SCHNEIDERMAN, B (1999): *Readings in Information Visualization – Using Vision to Think*, San Francisco, Morgan Kauffman Publishers.

FURNAS, G.W. (1981): The Fisheye View: A New Look at Structured Files, in *Readings in Information Visualization – Using Vision to Think*, S. Card et al, editors, San Francisco, Morgan Kauffman Publishers.

GEF.TIGRIS.ORG (2002): GEF Documentation,, http://gef.tigris.org/project_docs.html.

KÖTH, O, MINAS, M. (2001): Abstraction in graph-transformation based diagram , in *Electronic Notes in Theoretical Computer Science*, volume 50, L. Baresi, M. Pezze, G. Taentzer, editors, Elsevier Science Publishers.

LEUNG, Y.K., APPERLEY, M.D. (1994): A review and taxonomy of distortion-oriented presentation techniques, *ACM Transactions on Computer-Human Interaction*, **1**(2): 126-160.

OBJECT MANAGEMENT GROUP, INC (2000): Unified Modeling Language (UML) Specification Version 1.4, http://www.omg.org/technology/ documents/formal/uml.htm

PAIGE, R., OSTROFF, J., BROOKE, P., (2000): Principles for Modeling Language Design, *Information and Software Technology,* **42**(July 2002): 665-675.

RATIONAL SOFTWARE CORPORATION (2002): Rational Rose, http://www.rational.com/products/ rose/index.jsp.

WARE, C. (2000): *Information Visualization – Perception for Design*, San Diego, CA, Morgan Kauffman Publishers.

WICKENS, C.D. (1992): *Engineering Psychology and Human Performance, 2nd ed.*, Harper Collins, New York.